Introduction

As part of a new project, a client of ours wants to prototype a new semi-autonomous hoverboard concept, introducing an auto-pilot mode for easy though safe cruising over long distances. Based on this, the client has asked us to code a proof of concept to get the feeling on how this would work and to perform a demo to him with the MVP. If he is satisfied with the demo, a new long term contract will be signed to code the product from the ground up.

Based on the previous information, you have been appointed as the leader and main developer of the MVP; if you succeed with the demo, you will act as the PL of the new project. You will be in charge of doing a 30' demo showing the MVP.

Challenge

In order to quickly craft an MVP to test the client's idea, you are asked to build a prototype.

1. Use existing technologies and tools to speed up the process. In particular, we will base the prototype on ROS 2 Foxy running on Ubuntu 20.04. We will take advantage of Gazebo, starting out with the diff drive demo (see bottom section on how to run it). So the first step is to have the complete stack setup, and be able to teleop the robot.
2. Prototype the auto-pilot mode
    a. Cruising speed
        i. Implement a C++ ROS node to subscribe to a /user_vel ROS topic and forward messages to the /cmd_vel ROS topic. All teleop commands must now go through the /user_vel ROS topic.
        ii. Modify that C++ ROS node to expose /start_cruising and /stop_cruising ROS services. When cruising is started, the last commanded velocity will become the cruising speed, until it is stopped or the user attempts to change driving direction within some tolerance.
        iii. [BONUS] For unit testing the logic, gtest is going to be used.
        The unit test must not create any ROS entities, i.e. not services/subscriptions/etc, but must directly call the callbacks.
        Example test cases
            1. User vel is received, start cruising -> check cruising velocity
            2. User vel is received, start cruising, new velocity changing the driving direction -> check cruising is stopped and test the output velocity value.
            3. Think of other relevant corner cases.
    b. Go back home
        i. Implement a Python ROS node that records the base trajectory. It is up to you to define how those trajectories are sampled (e.g. fixed distance, fixed frequency, etc.).

ii. [BONUS] Extend that Python ROS node to expose a /go_home ROS service that will drive the base along the recorded trajectory in reverse i.e. back to the origin.

iii. [BONUS] Write unit tests using pytest.
Similarly than in the cpp case, we want to avoid creating ROS entities during the tests. Calling the subscription/service callbacks directly or using mocking utilities (https://docs.python.org/3/library/unittest.mock.html) are both acceptable.

3. [BONUS] Transition to a self-balancing prototype:
    a. Modify diff drive model. Remove casters and center the base. Then, add an IMU to the base.
    b. Modify diff drive plugin to close the loop using IMU measurements (i.e. base orientation) and maintain balance. Ensure your control loop behaves as the COM moves (hint: add a link to your model).

All BONUS points add up to the MVP but are not mandatory.
Priority order of bonus points: 2.a.iii, 2.b.ii, 2.b.iii, 3.

Deliverable

- The result must be provided in the form of a github repository provided by Ekumen, with clear instructions on how to build and execute the application as well as any dependencies required and tooling setup.
- The ROS node(s) should work on an Ubuntu 20.04 box.
- A 30' demo where we will show the MVP to our client.

General considerations

- We see this challenge as a holistic experience to evaluate how we would work together. For this reason communication and presentation are as important as the code itself, so we encourage you to stay connected with the team during the whole process.
- You can ask as many questions as you need and make any suggestions as you wish both at the beginning and throughout the challenge.
- After you receive all the information we will ask you to propose a delivery date within approximately 10 days of the time the task is presented to you. That time and date can be negotiated to accommodate your needs.
- The prototype does not have to function completely, although it will add to the evaluation.
- We will schedule a final interview to coincide right after your demo. During this second technical interview we will dig a bit deeper on the code and you will have an opportunity to tell us how you approached the challenge and any details that you feel are relevant to share with us outside of a demo.